The diagnostic capability of the Cray, gfortran, Intel, and Nag Fortran compilers: Revision 4

Ian D Chivers
Rhymney Consulting
Jane Sleightholme
FortranPlus

Revision history

The original version had entries for Cray, gfortran, Intel, Nag and Oracle.

Revision 1 dropped the Oracle entry as we were informed that development of the compiler had ceased and updated the Nag entry to the 7.0 release.

Revision 2 updated the Intel entry to the 2021.1 release and gfortran to the 10.2.0 release.

Revision 3 updated the Intel entries adding coverage of their new ifx compiler.

Revision 4 updated the Cray entry to cover the 10.0.4 release.

Introduction

We have used several compilers for the development of the examples in our books. The following table summarises this information:

We currently use the Nag, Intel, Cray and gfortran compilers.

Compiler diagnostic capability

The diagnostic capability of compilers is very important both in code development and when teaching Fortran.

Polyhedron Software and Services

`https://polyhedron.com/`

provide Fortran compiler comparison tables, and we have used their Fortran compiler diagnostic suite in testing the following releases

- Cray: 10.0.4
- gfortran: 10.2.0
- Intel ifort: 2021.1
- Intel ifx: 2021.1
- Nag: 7.0

Book(s)	Introducing Fortran 90	Introducing Fortran 95	Introduction to Programming with Fortran			
Year of publication	1995	2000	2006	2012	2015	2018
Edition			1	2	3	4
Compilers						
CVF	Yes	Yes	Yes			
Cray				Yes	Yes	Yes
DEC	Yes	Yes				
g95				Yes		
gfortran				Yes	Yes	Yes
IBM				Yes		
Intel			Yes	Yes	Yes	Yes
Lahey			Yes			
Nag	Yes	Yes	Yes	Yes	Yes	Yes
Oracle						Yes
Salford	Yes	Yes	Yes			
Sun				Yes	Yes	

Table 1: Compilers used

Source code exists for both Linux and Windows. The first table lists the source code test and run time error.

Table 2: Source code and run time error

Source	Run-time Error
A01, A02	Can mix checked and unchecked code
ARG1	argument mismatch: same file
ARG2	argument mismatch: different file6
ARG3	Illegal assignment to constant argument: no INTENT specified
ARG4	Illegal assignment to constant argument: INTENT specified
ARG5	Illegal assignment to DO loop variable in SUBROUTINE: no INTENT specified
ARG6	Illegal assignment to DO loop variable in SUBROUTINE: INTENT specified
ARG7	Scalar constant passed to array dummy argument
OARG1	Illegal use of optional argument
ALIAS1	Aliased dummy argument variable 10
ALIAS2	Aliased dummy argument array
BND1	array bound error: X(100)
BND2	array bound error: X(N) N is argument
BND3	array bound error: X(N) N is in COMMON
BND4	array bound error: X(N) N is in MODULE
BND5	array bound error: X(*)
BND6	array bound error: X(M:N) M and N argument question lower bound violated
BND7	array bound error: automatic array
	Continued on next page

Source	Run-time Error
BND8	array bound error: allocatable array
BND9	multi-dimensional array bound error within overall array bounds
BND10	array bound error: assign to dummy argument which is larger than actual argument
BND11	array bound error: pointer array component of derived type
CBND1	character bound error: local
CBND2	character bound error: COMMON
CBND3	character bound error: assign to dummy which is larger than actual argument
CBND4	character bound error: CHARACTER*(*)
UIN1	uninitialized variable7: local
UIN2	uninitialized variable: argument
UIN3	uninitialized variable: COMMON
UIN4	uninitialized variable: MODULE
UIN5	uninitialized array element: local
UIN6	uninitialized array element: argument
UIN7	uninitialized array element: COMMON
UIN8	uninitialized array element: MODULE
UIN9	uninitialized array element: local array in SUBROUTINE
UIN10	uninitialized array element: automatic arrays
UIN11	uninitialized array element: allocatable arrays
UIN12	uninitialized array element: saved arrays
UIN13	uninitialized array element: INTENT(OUT) arrays
UIN14	two byte array elements
UIN15	one byte array elements
DO1	zero increment DO loop
DO2	Illegal assignment to local DO loop variable
DO3	Illegal assignment to local DO loop variable via EQUIVALENCE
DO4	Illegal assignment to DO variables in CONTAINed subprogram
SF1	SUBROUTINE referenced as a FUNCTION: same file
SF2	SUBROUTINE referenced as a FUNCTION: different file6
FH1	Same file opened on 2 different units
FMT1	Illegal run-time format
CONF	Non-conformant array assignment
PTR1	Assign via pointer after target deallocated
PTR2	Assign via global pointer to local array after subprogram return
None	Full trace back from run-time errors

Here are the compiler switches used.

`cray`

```
ftn      -G 0 -K trap=denorm,divz,fp,inexact,inv,ovf,unf
         -O0 -R bcdps
```

`gfortran`

```
gfortran -W -Wall -fbounds-check -Wunderflow -O
         -fbacktrace -ffpe-trap=zero,overflow,underflow -g
```

```
Intel

  ifort    /check:all /fpe:0 /traceback /warn:all,nodec,interfaces
           /Qfp-stack-check  /gen_interfaces

  ifx      /check:all /fpe:0 /traceback /warn:all,nodec,interfaces
           /Qfp-stack-check  /gen_interfaces

NAG

  nagfor   -C=all -C=undefined -info
```

The next table has the diagnostic results for each compiler.

Table 3: Compiler test results

Source	Cray 10.0.4	gfortran 10.2.0	Intel ifort 2021.1	Intel ifx 2021.1	NAG 7.0
Score	49	53	55	28	96
ttft time	N/A, 30	2	10	7	58
A01, A02	Yes	Yes	Yes	Yes	Yes
ARG1	Yes 5	Yes 4	Yes 4	Yes 4	Yes 4
ARG2	Yes 5	No	Yes 4,8	Yes 4,8	Yes 5
ARG3	No	No	Yes 5,9	Yes 5,9	Yes 5
ARG4	Yes 4	Yes 4	Yes 4	Yes 4	Yes 4
ARG5	No	No	No	No	Yes 5
ARG6	Yes 4	Yes 4	Yes 4	Yes 4	Yes 4
ARG7	No	Yes 4	Yes 4	Yes 4	Yes 4
OARG1	Yes 5	Yes 4	Yes 5,9	Yes 5,9	Yes 5
ALIAS1	No	No	No	No	Yes
ALIAS2	No	No	No	No	No
BND1	Yes 5	Yes 5	Yes 5	No	Yes 5
BND2	Yes 5	Yes 5	Yes 5	No	Yes 5
BND3	Yes 5	Yes 5	Yes 5	No	Yes 5
BND4	Yes 5	Yes 5	Yes 5	No	Yes 5
BND5	No	No	No	No	Yes 5
BND6	Yes 5	Yes 5	Yes 5	No	Yes 5
BND7	Yes 5	Yes 5	Yes 5	No	Yes 5
BND8	Yes 5	Yes 5	Yes 5	No	Yes 5
BND9	Yes 5	Yes 5	Yes 5	No	Yes 5
BND10	No	No	No	No	Yes 5
BND11	Yes 5	Yes 5	Yes 5	No	Yes 5
					Contd:

Source	Cray 10.0.4	gfortran 10.2.0	Intel ifort 2021.1	Intel ifx 2021.1	NAG 7.0
Score	49	53	55	28	96
ttft time	N/A, 30	2	10	7	58
CBND1	Yes 5	Yes 5	Yes 5	No	Yes 5
CBND2	Yes 5	Yes 5	Yes 5	No	Yes 5
CBND3	No	Yes 5	Yes 4	Yes 4	Yes 4
CBND4	Yes 5	Yes 5	Yes 5	No	Yes 5
UIN1	No	Yes 3	Yes 5	No	Yes 3,5
UIN2	No	Yes 3	No	No	Yes 5
UIN3	No	No	No	No	Yes 5
UIN4	No	No	No	No	Yes 5
UIN5	No	Yes	No	No	Yes 5
UIN6	No	No	No	No	Yes 5
UIN7	No	No	No	No	Yes 5
UIN8	No	No	No	No	Yes 5
UIN9	No	No	No	No	Yes 5
UIN10	No	No	No	No	Yes 5
UIN11	No	No	No	No	Yes 5
UIN12	No	No	No	No	Yes 5
UIN13	No	No	No	No	Yes 5
UIN14	Yes 5	No	No	No	Yes 5
UIN15	Yes 5	No	No	No	Yes 5
DO1	Yes 5	No	DIV0 5,9	DIV0 5,9	Yes 5
DO2	Yes 4	Yes 4	Yes 4	Yes 4	Yes 4
DO3	No	No	No	No	Yes 4
DO4	No	No	No	No	Yes
SF1	Yes 4	Yes 4	Yes 4	Yes 4	Yes 4
SF2	No	No	Yes 4,8	Yes 4, 8	Yes 5
FH1	Yes 5	Yes 5	No	No	No
FMT1	Yes 5	Yes 5	Yes 5	Yes 5	Yes 5
CONF	Yes 3	Yes 5	Yes	No	Yes 5
PTR1	No	No	No	No	Yes 5
PTR2	No	Yes	No	No	Yes 5
UFL	Option	Option	Option	Option	Option
OFL	Option	Option	Option	Option	Option
DIV0	Option	Option	Option	Option	Option
IOFL	No	Silent	Silent	Silent	Option
None	No	No	Yes	Yes	Yes

Notes

- 1 Percentage Passes is calculated from a total of 53 tests. No score is assigned for exception handling.
- 2 Execution Time for TFFT.for, compiled using the diagnostic switches (see table above) measured on the same machine as the benchmarks.
- 3 Compiler issued warning message.
- 4 Compiler issued fatal error message no executable produced.
- 5 Run-time message.
- 6 Arranged so that error is not visible at compile time.
- 7 A variable is uninitialized if it has never been assigned a value. Data elements should be initialized before their value is used. The illegal use of an uninitialized variable can sometimes be detected at compile-time, but, because it may be data and flow dependent, usually requires run-time monitoring. In some cases, compilers implement this by checking for a special uninitialized bit pattern, such as 0x80, but this may result in false positives for one or two byte variables (and, rarely, in other cases). Use of a shadow array doubles memory requirements, but avoids false positives.
- 8 Compilation order dependent. Compiler generates .mod file to allow interface checking.
- 9 Non-specific error message, but full traceback.
- 10 Spotted by Forcheck.
- 11 The test is passed if the program refuses the second open. For some reason, the Fortran standard does not allow a file to be opened twice, even for read-only access. It has been argued to me that allowing the second open is an extension to the standard, rather than a failure to spot an error.
- 12 False positive detected
- 20 compiler option -ftrap=overflow,underflow. Has no effect
- 30 The Cray compiler has a N/A entry for tfft execution time as the timing for this compiler was done using the UK HPC systems at Edinburgh. The other timing figures are on the same system.

Summary

The Nag compiler has the highest diagnostic rating using the Polyhedron code suite.

Acknowledgements

We would like to thank Polyhedron Software and Services for allowing us to use their diagnostic suite.

